UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE TELECOMUNICAÇÕES

CFAR para detecção dos picos de cada alvo na superfície ARD e filtros de multiple hits

PATRICIA TOMALAK LISS, RAFAEL ZIANI DE CARVALHO

PROJETO INTEGRADOR II
Fernando César Comparsi de Castro e Natanael Rodrigues

Santa Maria, RS
23 de julho de 2024

Sumário

1 Introdução

O radar passivo bistático é uma tecnologia que utiliza transmissões de rádio já existentes no ambiente para detectar e localizar objetos, sem a necessidade de um transmissor próprio. Diferente dos radares monostáticos, que têm transmissor e receptor no mesmo local, o radar bistático posiciona esses componentes em locais separados, permitindo novas configurações operacionais. [1]

Também conhecido como radar "verde"ou "ecológico", o radar passivo aproveita sinais presentes no ambiente, originalmente gerados para outras finalidades, eliminando a necessidade de novos transmissores e evitando interferências eletromagnéticas. Essa característica torna o radar passivo uma solução eficiente e sustentável. [1]

Pode-se dizer que, os radares passivos são frequentemente implementados usando o conceito de rádio definido por software, realizando a maior parte do processamento de sinais digitalmente em banda-base. Esse processamento inclui a filtragem adaptativa para minimizar a interferência do sinal do caminho direto (DPI) e o uso de funções de ambiguidade ou processamento range-Doppler para correlacionar o sinal de referência com os ecos recebidos dos alvos. [1]

Para melhor entendimento das subdivisões que compõem o desenvolvimento de um radar passivo biostático, temos:

- Filtragem adaptativa para minimização da interferência do sinal do caminho direto, que consiste na interferência do sinal de referência residual sobre o sinal dos ecos do(s) alvo(s) recebidos nos canais de vigilância – Direct Path Interference (DPI).
- Função de ambiguidade (matched-filter) ou processamento range-Doppler através da correlação cruzada entre o sinal de referência e o sinal dos ecos do(s) alvo(s) recebidos nos canais de vigilância.
- Detecção do(s) alvo(s) através do algoritmo Constant False Alarm Rate (CFAR) e minimização da multiplicidade de hits em situação de recepção adversa – Multiple Hits Filters.
- Localização do(s) alvo(s) a partir da amplitude e fase das ondas EM recebidas nos canais de vigilância.

Para a detecção precisa de alvos, um dos algoritmos mais utilizados é o Constant False Alarm Rate (CFAR), que garante uma taxa constante de alarmes falsos ao ajustar dinamicamente o limiar de detecção com base no ruído de fundo ao redor da célula sob teste (CUT). O presente trabalho vai se aprofundar no desenvolvimento do algoritmo CFAR.

2 Constant False Alarm Rate (CFAR)

O Constant False Alarm Rate (CFAR) é uma técnica utilizada em sistemas de radar para a detecção de alvos. O principal objetivo do CFAR é garantir que a taxa de alarmes falsos permaneça constante, independentemente das condições variáveis do ambiente e do ruído de fundo. Isso é conseguido ajustando dinamicamente o limiar de detecção com base no ruído de fundo ao redor da célula sob teste (CUT). [1]

Podemos iniciar discutindo cada tópico dentro do algoritmo de detecção CFAR e analisar suas funcionalidades, sendo assim:

2.0.1 Matriz de Entrada

Inicialmente, deve-se entender que há a matriz de entrada, que é representada por $\Psi(\text{range}, \nu)$, sendo ela uma matriz de valores complexos de tamanho $[N_{\text{Doppler}}, N_{\text{Range}}]$. Para simplificação, podemos considerar range $= x$ e $\nu = y$, então $\Psi(x, y) = \Psi(\text{range}, \nu)$. A Fig. 1 ilustra a matriz que será gerada de tamanho NumDoplerF e NumRangePoints.

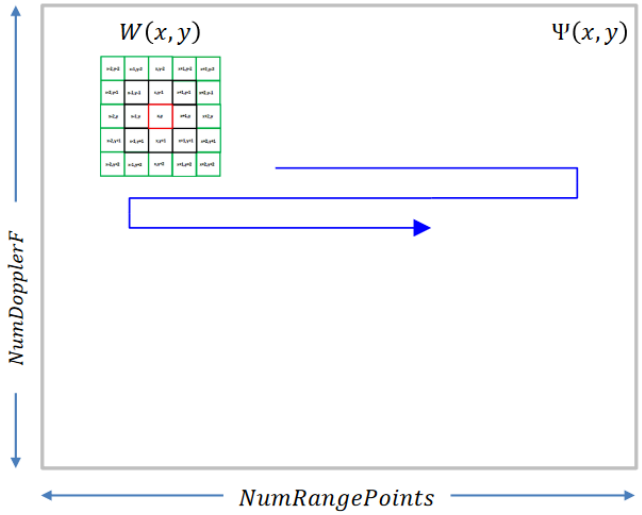


Figura 1: Matriz de entrada $\Psi(\text{range}, \nu)$.

ssa matriz de entrada será posteriormente chamada de matriz de magnitude, ou simplesmente mag, e ela representa a magnitude de cada posição na matriz correspondente às dimensões de range e Doppler. Através dessa matriz, podemos obter os valores que serão usados para comparação, permitindo a detecção de alvos. Dessa forma, é possível identificar as posições onde existem alvos dentro da matriz mag.

2.0.2 Janela $W(x, y)$

Após isso, podemos adentrar a janela $W(x, y)$, sendo essa uma submatriz de $\Psi(x, y)$ de tamanho [WindowSize, WindowSize]. Esta janela move-se da esquerda para a direita e de cima para baixo, percorrendo todo o domínio (x, y) de Ψ. Vale ressaltar que a janela também está ilustrada na Fig. 1, com a Fig. 2 podemos visualizar melhor a nossa submatriz.

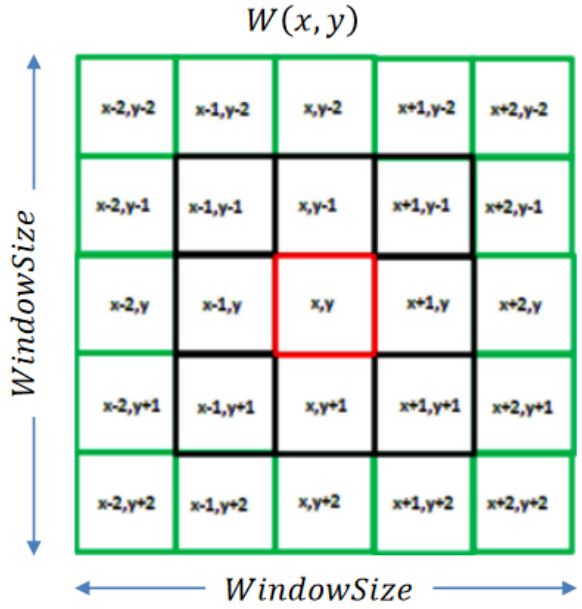


Figura 2: Submatriz W(x,y).

Essa submatriz terá a função de percorrer nossa matriz mag, e ela poderá analisar para cada posição da célula vermelha no centro de W(x,y), ao longo do movimento de W no domínio (x,y), Assim, CFAR testa a magnitude, da célula vermelha W(x,y), compara com a média dos valores de magnitude das células adjacentes verdes e toma a decisão se há ou não um alvo na posição (x,y), da célula vermelha. Uma análise mais aprofundada será discutida posteriormente.

A submatriz $W(x, y)$ percorre a matriz principal de magnitude $\Psi(x, y)$, analisando a magnitude $|\Psi(x, y)|$ da célula sob teste (CUT) em cada posição. Para cada posição na janela $W(x, y)$, a célula vermelha no centro é testada através de uma comparação.

2.0.3 Cálculo da Média das Células de Referência

Para realizar a detecção, é necessário calcular a média μ das magnitudes das células de referência (células em verde). Esta média será usada para estabelecer um limiar de comparação entre a célula sob teste (CUT) e a média:

$$\mu = \frac{1}{N_\Delta} \sum_{x_R} \sum_{y_R} |\Psi(x_R, y_R)|$$

onde N_Δ é o número de células de referência, e (x_R, y_R) são as posições das células de referência em $W(x, y)$.

2.0.4 Teste de Detecção

A decisão de detecção é feita comparando a magnitude $|\Psi(x, y)|$ da célula sob teste (CUT) com um limiar T, que é a média μ multiplicada por um fator de escalamento (Threshold):

$$T = \mu \cdot \text{Threshold}$$

Se $|\Psi(x, y)| > T$, um alvo é detectado na posição (x, y). O limiar (Threshold) define o nível de comparação entre a célula sob teste (CUT) e a média das células de referência, sendo crucial para decidir se um alvo está presente em uma determinada posição da matriz de dados.

2.0.5 Filtros de Hits Múltiplos

Após a detecção, o algoritmo CFAR pode identificar múltiplos hits em torno de um único pico devido às declividades moderadas.

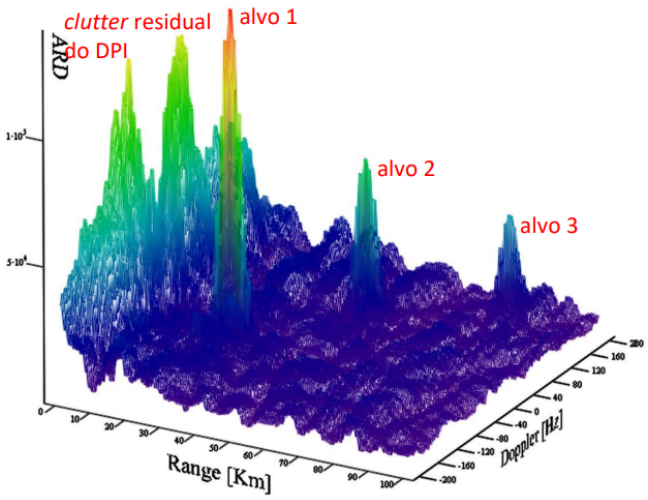


Figura 3: Alvos com suas declividades moderadas

Para melhorar a precisão da detecção e reduzir a redundância, são utilizados filtros para eliminar esses múltiplos hits. Abaixo, são descritos os principais filtros aplicados:

- **MultipleRangeHitFilter e MultipleDopplerHitFilter**: Estes filtros são projetados para lidar com múltiplos hits em posições de índice iguais no domínio do alcance e Doppler, respectivamente. O princípio básico é que, para cada posição de índice específica, o filtro retém apenas o hit com a maior magnitude $\Psi(x, y)$ e descarta os demais. Isso se baseia na suposição de que o hit com maior magnitude representa o alvo mais significativo, enquanto os hits menores são considerados como ruído ou interferência.

- **AdjacentRangeHitFilter e AdjacentDopplerHitFilter**: Estes filtros são aplicados para lidar com hits em posições de índice consecutivos no domínio do alcance e Doppler, respectivamente. O filtro retém o hit com a maior magnitude $\Psi(x, y)$ em posições adjacentes e descarta os demais. A ideia é que hits em posições adjacentes podem ser parte do mesmo alvo ou estrutura e, portanto, o filtro mantém o hit mais proeminente para evitar a fragmentação dos alvos e reduzir o efeito de múltiplos hits que podem ocorrer devido a variações menores na declividade do pico.

Esses filtros são essenciais para garantir que a detecção final represente com precisão os alvos reais e minimize a interferência dos múltiplos hits causados pelas declividades moderadas e pela natureza do sinal iluminante.

3 Desenvolvimento

Com a base teórica revisada, podemos começar a implementação do algoritmo solicitado neste relatório:

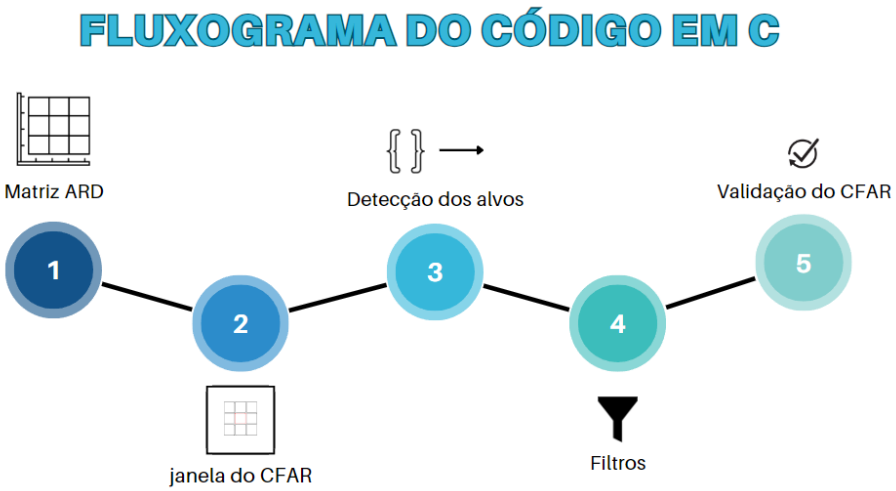


Figura 4: Fluxograma do algoritmo em C

Podemos analisar através do fluxograma, que para implementarmos corretamente o CFAR, devemos realizar 4 etapas primordiais, sendo elas a criação da nossa matriz mag, a criação da nossa submatriz, o algoritmo responsável por movimentar a submatriz e obter os alvos e por fim seus respectivos filtros, assim completando o algoritmo solicitado e validando o CFAR.

3.1 matriz mag

A primeira etapa do algoritmo é obter a matriz de magnitude, denominada matriz mag. Para isso, devemos utilizar o arquivo base Mathcad disponibilizado pelos professores. Primeiro, acesse o software Mathcad e gere a matriz de magnitude a partir do arquivo base. Em seguida, será necessário implementar funções em C para analisar o arquivo de texto (.txt) gerado pelo Mathcad. Essas funções serão responsáveis pela leitura e impressão da matriz a partir do arquivo, garantindo que a matriz de magnitude esteja corretamente formatada e disponível para análise subsequente no algoritmo.

Após isso, podemos então começar a implementação dos códigos em C para a leitura e impressão desse arquivo de texto:

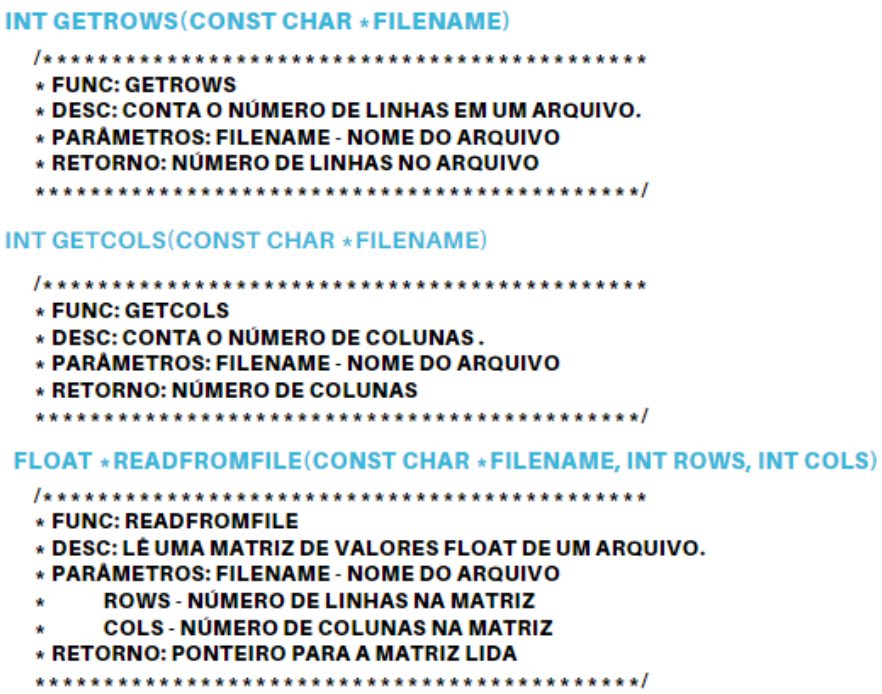


Figura 5: pseudocódigos da matriz mag

Para que a leitura seja realizada de maneira eficiente, utilizaremos três funções principais e uma função auxiliar, a função GETROWS tem o objetivo de adentrar o arquivo txt e ler o numero de linhas do arquivo, a função GETCOLS tem a função de contar o numero de colunas desse mesmo arquivo, e por fim a função READFROMFILE irá ler os valores dessa matriz.

Figura 6: pseudocódigo para a função auxiliar da matriz mag

a função auxiliar terá o objetivo de imprimir a matriz mag após a leitura de linhas, colunas e valores.

com todas as funções apresentadas, podemos fazer a impressão da matriz mag pelo C e também podemos abrir o proprio arquivo txt para assim compararmos os resultados obtidos:

```
/*********************************************************
* MAIN():
*********************************************************/
INT MAIN() {
  CONST CHAR *FILENAME = "MAG.TXT";
  INT ROWS = GETROWS(FILENAME); // ARMAZENA O NÚMERO DE LINHAS DA MATRIZ
  INT COLS = GETCOLS(FILENAME); // ARMAZENA O NÚMERO DE COLUNAS DA MATRIZ
  FLOAT *MATRIX = READFROMFILE(FILENAME, ROWS, COLS); // FUNÇÃO READFROMFILE E LÊ A MATRIZ DO ARQUIVO

  // 1. MATRIZ MAG
 #IF 1
 PRINTF("MATRIX (%D X %D):\N", ROWS, COLS);
 PRINTMATRIX(MATRIX, ROWS, COLS);
 #ENDIF
```

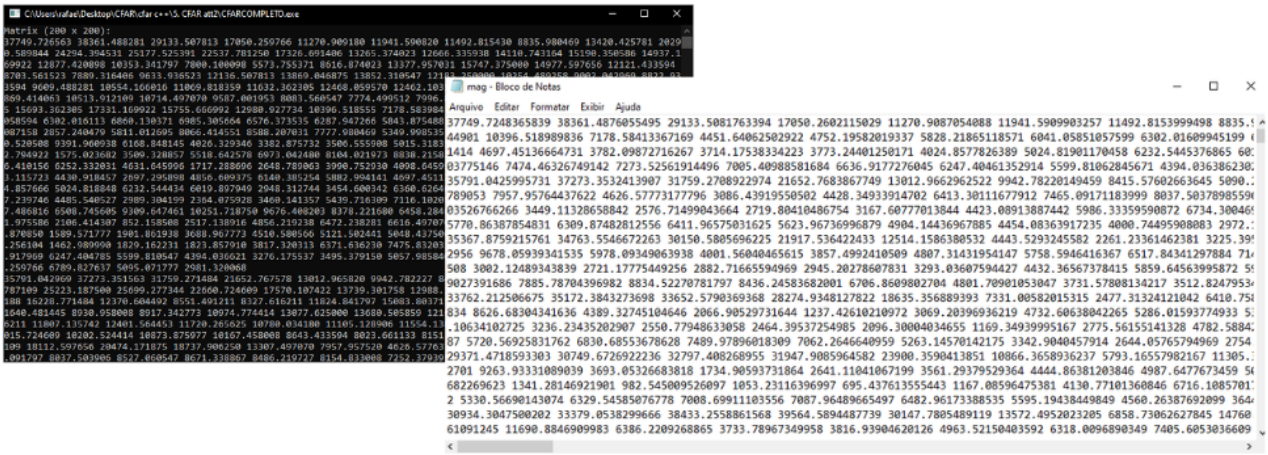


Figura 7: Arquivo txt da matriz mag em comparação ao código e impressão da matriz mag

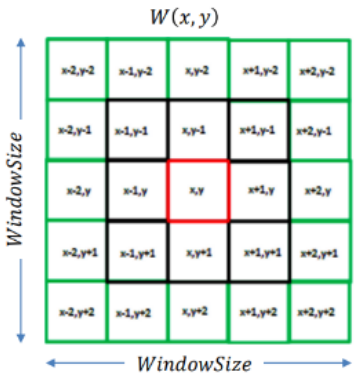
Com isso, podemos analisar através dos resultados, que conseguimos imprimir com sucesso a nossa matriz mag, e ela está coerente com os valores do proprio arquivo de texto obtido do mathcad, assim validando a primeira etapa do CFAR.

3.2 Janela do CFAR

Nosso próximo objetivo, é imprimir uma janela para o CFAR:

JANELA DO CFAR

```
INT** CFARWINDOW(INT WINDOWSIZE, INT XFLAG, INT* VETORSIZE)
/*****************************************
* FUNC: CFARWINDOW
* ----------
* DESC: CRIA UMA JANELA CFAR (CONSTANT FALSE ALARM RATE) COM BASE NO TAMANHO FORNECIDO.
*
* PARÂMETROS: WINDOWSIZE: O TAMANHO DA JANELA, QUE DEVE SER UM NÚMERO ÍMPAR ENTRE 5 E 8192.
* XFLAG: UM INDICADOR UTILIZADO PARA DETERMINAR O TIPO DE OPERAÇÃO:
*     - XFLAG = -2: RETORNA A MATRIZ TRANSPOSTA DE Δ ONDE AS CÉLULAS DE REFERÊNCIA ARMAZENAM VALORES DE DESLOCAMENTO DELTA DO ÍNDICE Y.
*           AS POSIÇÕES CORRESPONDENTES À CUT E ÀS CÉLULAS DO ANEL DE GUARDA ARMAZENAM O VALOR 8192.
*     - XFLAG = -1: RETORNA A MATRIZ Δ ONDE AS CÉLULAS DE REFERÊNCIA ARMAZENAM VALORES DE DESLOCAMENTO DELTA DO ÍNDICE X.
*           AS POSIÇÕES CORRESPONDENTES À CUT E ÀS CÉLULAS DO ANEL DE GUARDA ARMAZENAM O VALOR 8192.
*     - XFLAG = 1: TRANSFORMA A MATRIZ Δ EM UM VETOR E E RETORNA OOS VALORES DO VETOR (AS CÉLULAS 8192 DESAPARECEM).
*     - XFLAG = 0: TRANSFORMA A MATRIZ TRANSPOSTA DE Δ EM UM VETOR E RETORNA OS VALORES DO VETOR TRANSPOSTO (AS CÉLULAS 8192 DESAPARECEM).
* VETORSIZE: PONTEIRO PARA UM INTEIRO QUE SERÁ PREENCHIDO COM O TAMANHO DO VETOR RETORNADO(ESTE PARÂMETRO SÓ É UTILIZADO QUANDO XFLAG É 0 OU 1).
*       INDICA O NÚMERO DE ELEMENTOS VÁLIDOS (DIFERENTES DE 8192) NO VETOR RETORNADO.
*****************************************/
```



FUNÇÕES AUXILIARES

```
/*****************************************
* FUNCTION: IMPRIMIRXFLAG_MINUS1
* ----------
* DESC: IMPRIME A JANELA PARA XFLAG = -1, RETORNANDO A MATRIZ Δ

/*****************************************
* FUNCTION: IMPRIMIRXFLAG_MINUS2
* ----------
* DESC: IMPRIME A JANELA PARA XFLAG = -2, RETORNANDO A MATRIZ Δ TRANSPOSTA

/*****************************************
* FUNCTION: IMPRIMIRXFLAG_1
* ----------
* DESC: XFLAG = 1, TRANSFORMA A MATRIZ Δ EM UM VETOR E IMPRIME O VETOR

/*****************************************
* FUNCTION: IMPRIMIRXFLAG_0
* ----------
* DESC: XFLAG = 0, TRANSFORMA A MATRIZ Δ TRANSPOSTA EM UM VETOR E IMPRIME O VETOR

/*****************************************
* FUNCTION: PRINTVETOR
* ----------
* DESC: IMPRIME OS VALORES DE UM VETOR.
```

Figura 8: pseucódigos para a janela do CFAR

Com tais funções implementadas, podemos então tentar aplicar uma comparação através da janela obtida pelo mathcad em um algoritmo já implementado de CFAR, sendo assim, pegaremos os nossos resultados e imprimiremos da mesma forma que está exposta no mathcad, para assim validar a nossa janela CFAR, utilizaremos inicialmente um windowsize = 5:

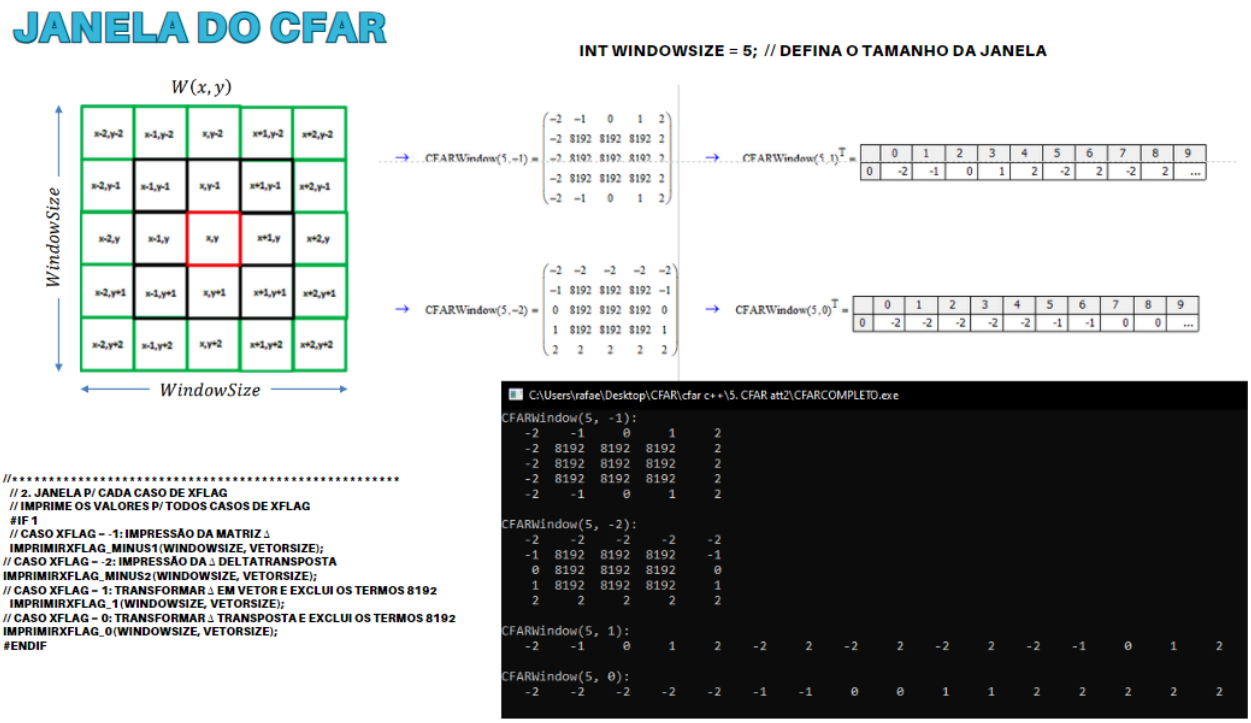


Figura 9: Testando a janela do CFAR para Windowsize = 5

Como é possível analisar, ao imprimirmos Delta, Delta transposto, vetor de delta e vetor de delta transposto, obtemos os mesmos resultados apresentados no mathcad de referência, tal fato comprova que a janela está sendo criada da maneira correta, imprimindo os valores de delta e delta transposto corretamente e ao trabalhar com os valores dos vetores desses Deltas ele desconsidera os valores da CUT e do anel de guarda.

Iremos fazer mais um teste para comprovar que a janela está se adaptando também de acordo com nosso valor de Windowsize, assim sendo, mudaremos o valor de windowsize de 5 para 7:

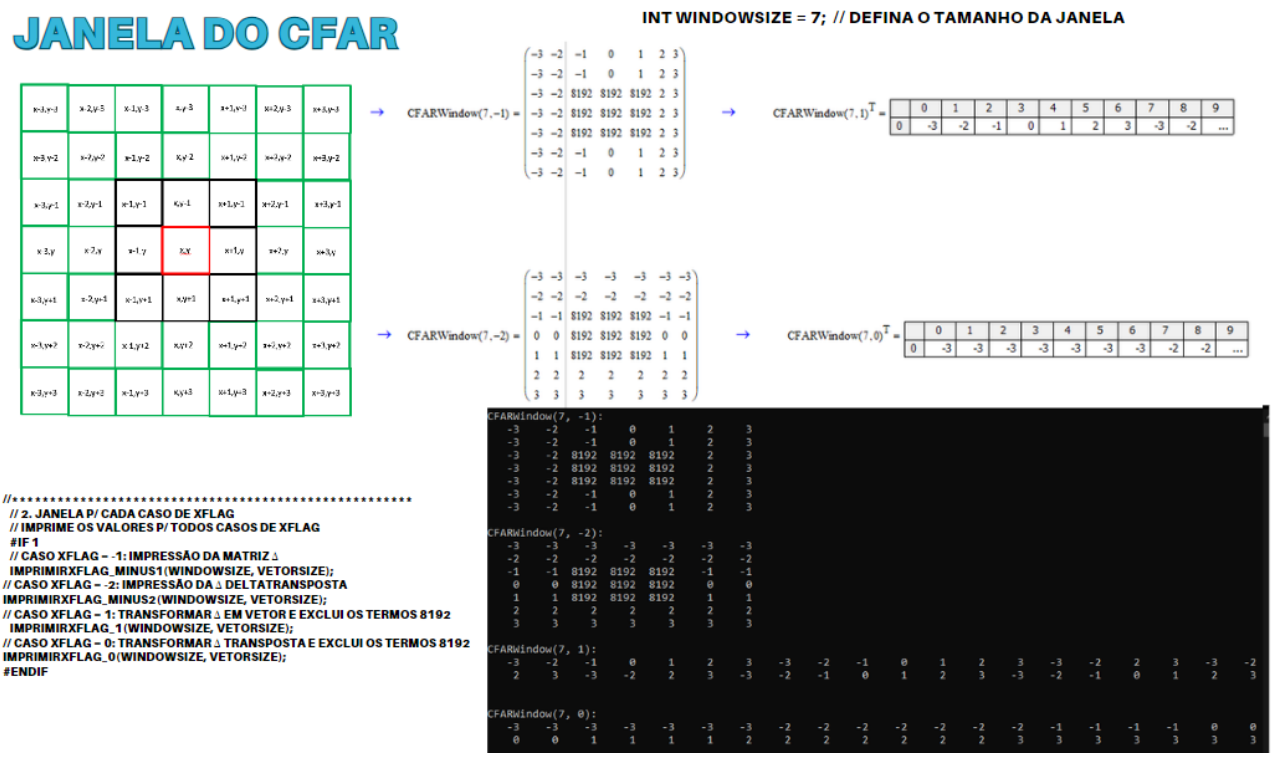


Figura 10: Testando a janela do CFAR para Windowsize = 7

Com isso, podemos então comprovar alterando do valor de Windowsize obtemos os resultados esperados, se analisarmos o mathcad, podemos ver que igualmente para cada caso de xflag obtemos determinado resultado, e isso também ocorre dentro do algoritmo implementado, em relação ao windowsize, podemos tambem notar que o valor dele precisa seguir algumas regras: precisa ser um número impar, igual ou maior que 5 e menor ou igual 8192, a janela poderá se adaptar a todos esses valores dentro dessas restrições.

Por conseguinte, tendo a nossa janela do CFAR implementada, podemos então seguir para o proximo passo, sendo onde a detecção dos alvos ocorre.

3.3 Detecção dos alvos

Para implementarmos a detecção dos alvos, devemos implementar o movimento da nossa janela do CFAR sobre a nossa matriz mag, assim, devemos também consideran o critério de avaliação e comparação entre a média das das celulas sobre teste e celulas de referência.

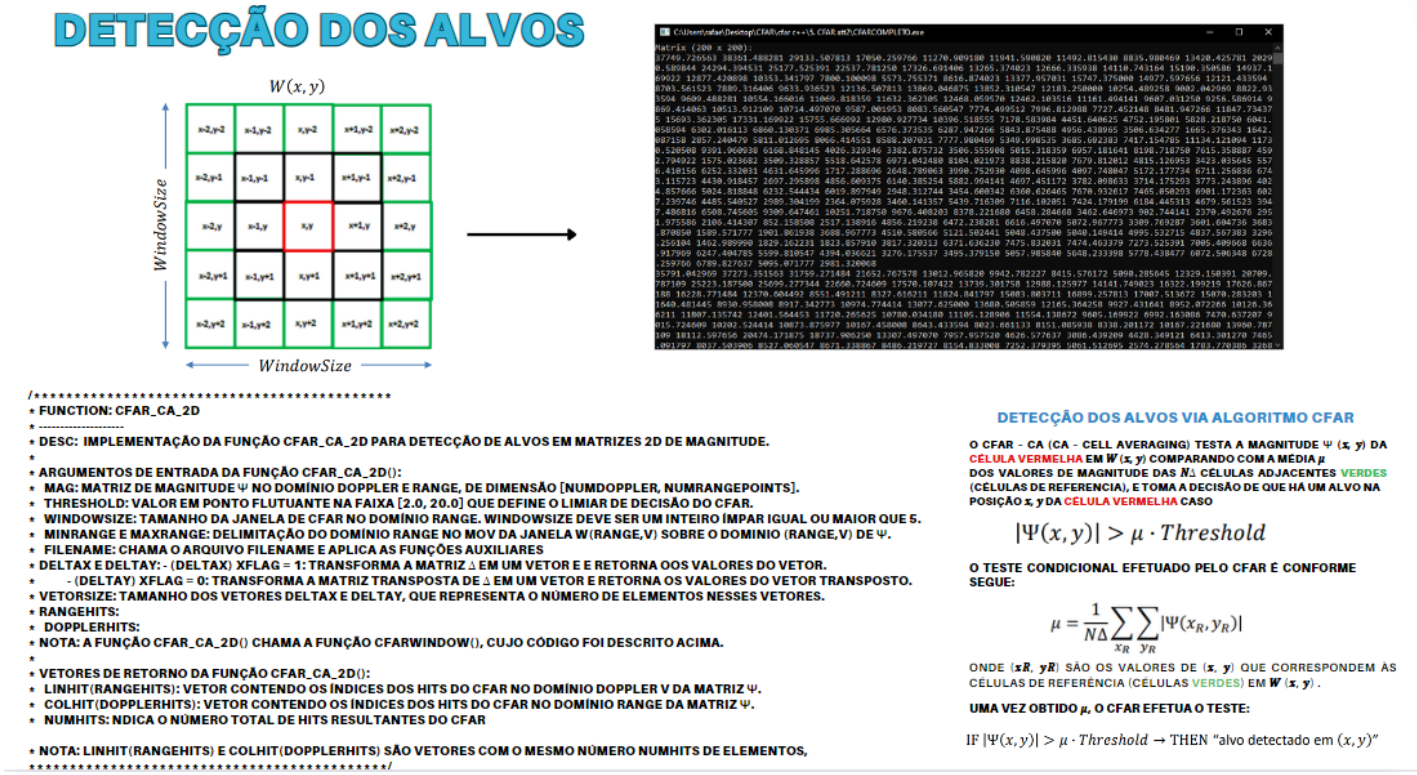


Figura 11: Algoritmo para detecção dos alvos

Com isso, oque buscamos é uma função para detecção dos alvos através dos critérios expostos na detecção dos alvos via algoritmo do CFAR, exposto na figura acima, tendo em vista que o algoritmo foi implementado, devemos imprimir resultados para windowsize = 5 e analisar oque foi obtido:

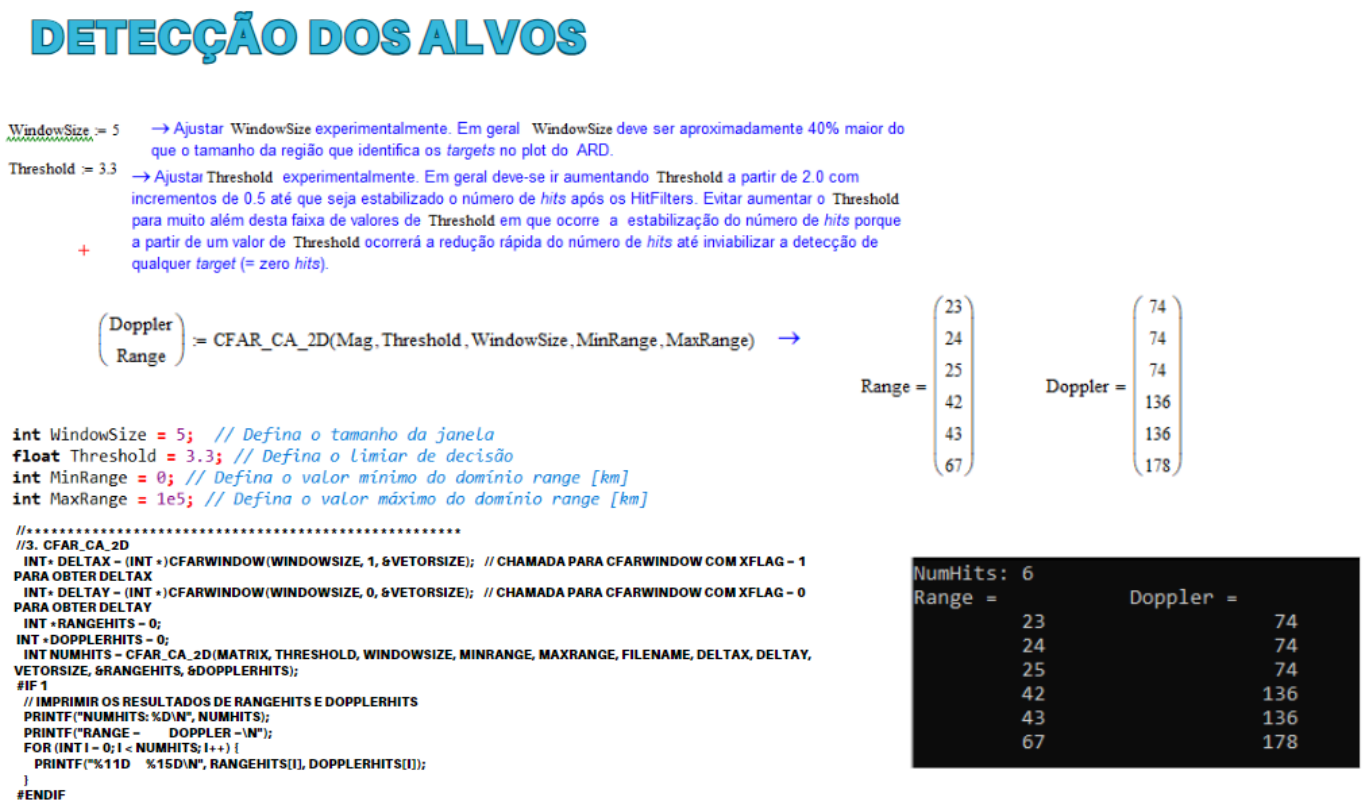
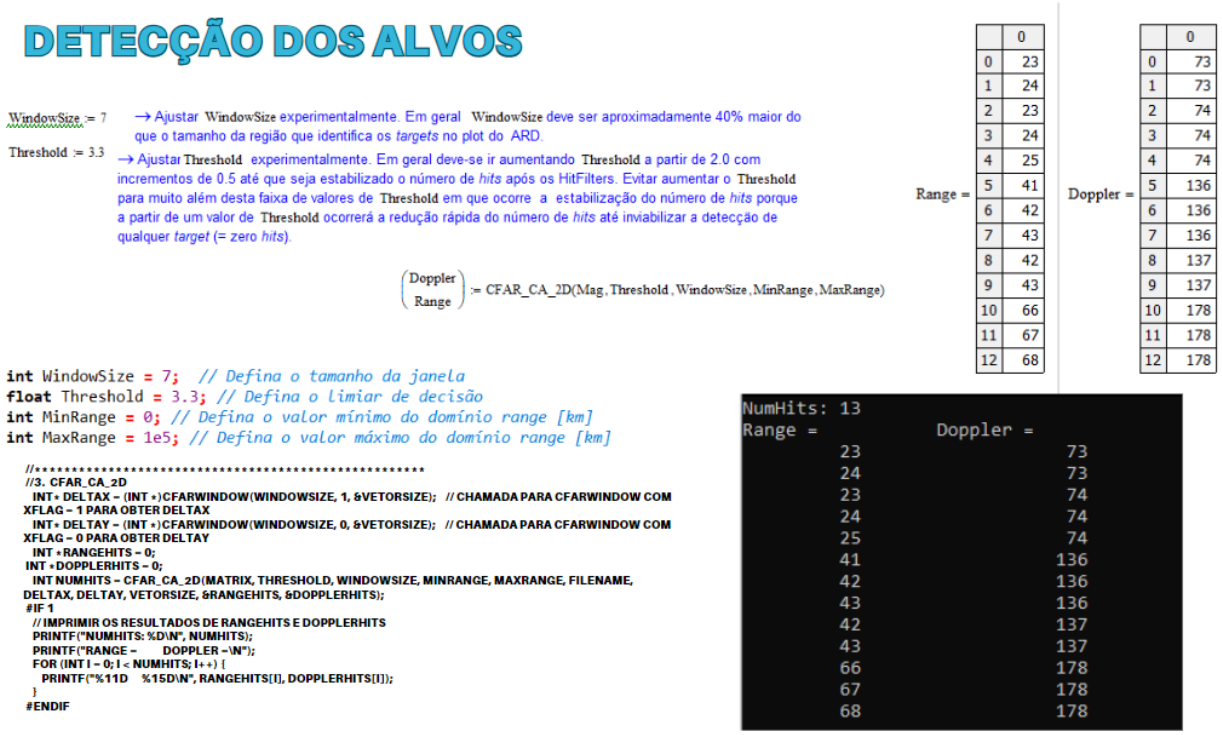


Figura 12: Detecção dos alvos para Windowsize = 5

Implementando o algoritmo para uma janela de tamanho = 5, podemos notar que ao compararmos com os resultados obtidos no mathcad, temos os mesmos valores obtidos dentro do mathcad, tal fato comprova que a detecção dos alvos está sendo bem implementada e de maneira eficiente, obtivemos o mesmo resultado que o mathcad de referência, iremos agora testar para uma windowsize = 7: Podemos observar também que há a impressão da quantidade de alvos detectados, sendo esse valor igual tanto para Range quanto para Doppler.



	0
0	23
1	24
2	23
3	24
4	25
5	41
6	42
7	43
8	42
9	43
10	66
11	67
12	68

	0
0	73
1	73
2	74
3	74
4	74
5	136
6	136
7	136
8	137
9	137
10	178
11	178
12	178

Figura 13: Detecção dos alvos para Windowsize = 7

Notamos que alterar o tamanho da janela altera os resultados, porém, assim como esperado, a nossa detecção dos alvos também se adapta as mais variadas condições, as funções implementadas podem ser analisadas detalhadamente através das imagens ou ao fim do relatório, no código anexado.

Devemos também ressaltar que o valor de threshold está fixado em 3.3 em ambos os códigos, seja no mathcad ou no algoritmo em C, Mas podemos também definir um limiar diferente de acordo com o necessário para cada situação, um possível procedimento heurístico para determinar um ótimo threshold para o algoritmo CFAR-CA consiste em aumentar gradualmente o threshold a partir de 2.0 até que o número de hits pós-filtragem estabilize em um valor NumHits no centro da maior faixa de variação de threshold ao longo da qual NumHits é constante. Note que o número de hits pós-filtragem não é o número de hits retornados pela função CFAR CA 2D() mas sim é o número de hits após o conjunto de índices retornados pela função CFAR CA 2D() ser filtrado por MultipleRangeHitFilter(), MultipleDopplerHitFilter(), AdjacentRangeHitFilter() e AdjacentDopplerHitFilter()

3.4 Filtros

Por fim, temos ultima parte para comprovar o algortimo realizado em CFAR de acordo com o fluxograma apresentado.

os resultados obtidos anteriormente, representam os alvos acertados pré filtragem, isso é, o CFAR resultará em múltiplos hits nas vizinhanças do máximo local da superfície mag(Range, doppler) de cada alvo. Desta forma, poderão ocorrer múltiplos hits tanto no domínio range como no domínio Doppler v, assim como também poderão ocorrer hits em posições adjacentes de índices consecutivos tanto no domínio Range como no domínio Doppler v.

Para lidar com isso, deve-se implementar os algorítimos responsáveis pelas filtragens dos resultados

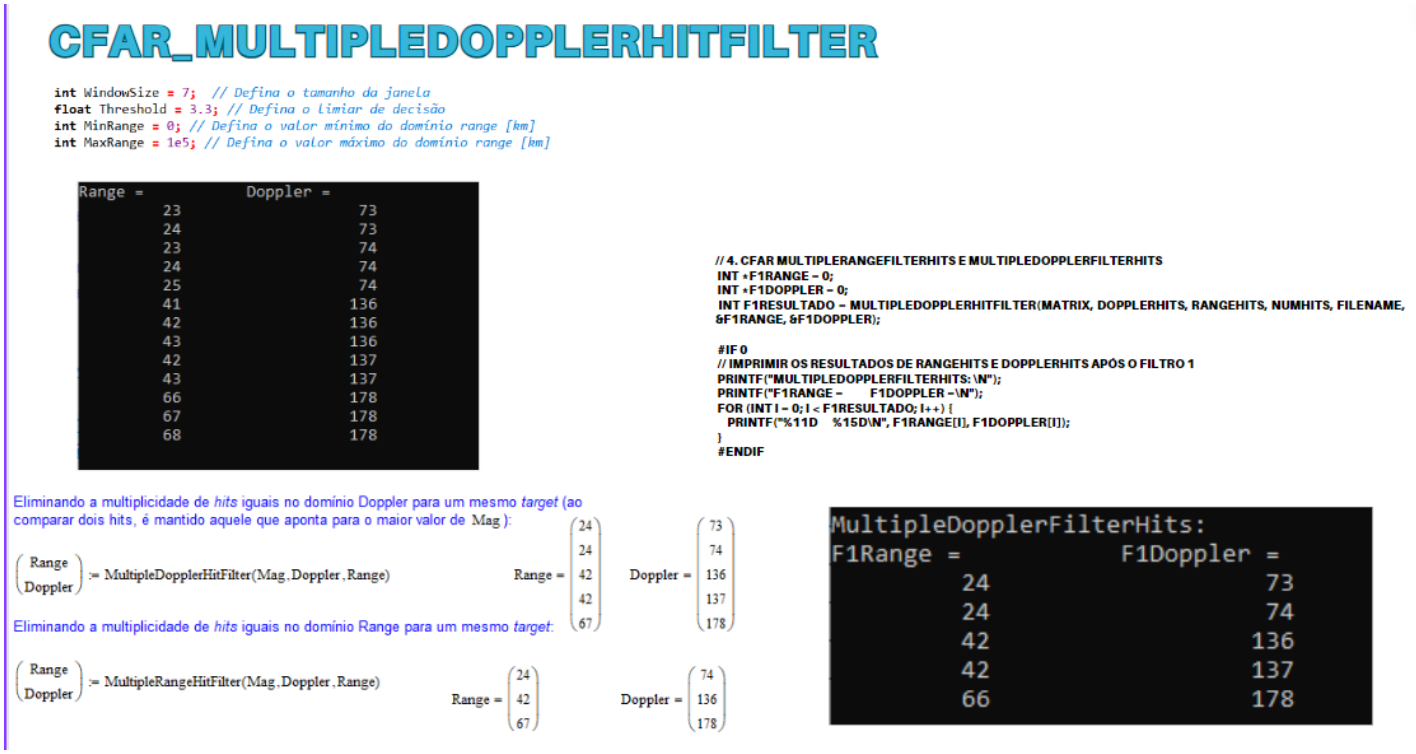


Figura 14: Filtro 1: MultipleDopplerRange

Podemos analisar que aplicando o MultipleDopplerHitFilter para um windowsize = 7 e um threshold = 3.3, obtemos então os valores filtrados de Doppler para os multiplos hits no dominio doppler.

Com isso, terminamos todas as funções que deram certo no algoritmo do CFAR, todos os códigos estão implícitos para este relatório a primórdio, porém a seguir evidenciaremos todos nossos códigos passo a passo, com seus respectivos comentários e verificações necessárias para uma boa organização.

4 Código Final

```
1
2  /*****************************************************
3  * HEADERS
4  *****************************************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <math.h>
8
9  /*****************************************************
10 * PROGRAM DEFINITIONS
11 *****************************************************/
12 #define T 4e-6 // intervalo de tempo entre s mbolos IQ adjacentes no tempo
13 #define C 2.998e8 // velocidade da luz em metros por segundo
14
15 /*
       ***************************************************************************
16 * DATA TYPE STRUCTURES:
17 ***************************************************************************
       */
18 typedef enum {
19     NO_ERROR,             /* "No error" */
20   MEMORY_ERR,           /* "Not enough memory" */
21   READ_OPEN_ERR,        /* "Unable to open file for reading" */
22   READ_ERR,             /* "Error reading file" */
23   WRITE_OPEN_ERR,       /* "Unable to open file for writing" */
24   WRITE_ERR,            /* "Error writing to file" */
25   BUFFER_ERR,           /* "Unable to attach buffer to file" */
26   SIZE_ERR,             /* "Specified size doesn't match data size in file" */
27     MAXIT_ERR,            /* "Exceeded maximum number of iterations in
       function" */
28     LOGIC_ERR,            /* "Logic error in function" */
29     CMDLINE_ERR,          /* "Incorrect command line argument" */
30     MTXDIM_ERR,           /* "Invalid matrix dimensions in file" */
31     NROWS_ERR,            /* "Incompatible number of rows between input
       matrices" */
32     NCOLS_ERR,            /* "Incompatible number of columns between input
       matrices" */
33     UNKNOWN_ERR           /* "Unknown error" */
34 } ERR;
35
36 /*****************************************************
37 * FUNCTION PROTOTYPES
38 *****************************************************/
39 void Quit(ERR err, const char *name);
40
41 /*********************************
42 * PROTOTIPOS DAS FUN   ES MATRIZ_ARD
43 *********************************/
44 int getRows(const char *filename);
45 int getCols(const char *filename);
46 float *readFromFile(const char *filename, int rows, int cols);
47 void printMatrix(float *matrix, int rows, int cols);
48
49 /*********************************
50 * PROTOTIPOS DAS FUN   ES CFARWindow
51 *********************************/
```

```
52 int** CFARWindow(int WindowSize, int xFlag, int* vetorSize);
53 void printVetor(int* vetor, int size);
54 void imprimirxFlag_minus1(int WindowSize, int vetorSize);
55 void imprimirxFlag_minus2(int WindowSize, int vetorSize);
56 void imprimirxFlag_1(int WindowSize, int vetorSize);
57 void imprimirxFlag_0(int WindowSize, int vetorSize);
58
59 /*************************************
60 *** PROT TIPO DA FUN   O CFAR_CA_2D
61 *************************************/
62 int CFAR_CA_2D(float *mag, float Threshold, int WindowSize, int MinRange,
       int MaxRange, const char *filename, int* deltax, int* deltay, int
       vetorSize, int **LinHit, int **ColHit);
63
64
65 /*
       ***************************************************************************

66 *** PROT TIPOS DAS FUN   ES MULTIPLERANGEHITFILTER E
       MULTIPLEDOPPLERHITFILTER
67 ***************************************************************************
       */
68 int MultipleDopplerHitFilter(float *mag, int *DopplerHits, int *RangeHits,
       int numHits, const char *filename, int **F1Range, int **F1Doppler);
69 //int MultipleRangeHitFilter(float *mag, int *DopplerHits, int *RangeHits,
       int numHits, const char *filename, int **F2Range, int **F2Doppler);
70
71
72 /*********************************************************
73 * MAIN():
74 *********************************************************/
75 int main() {
76     const char *filename = "mag.txt";
77     int rows = getRows(filename); // armazena o n mero de linhas da matriz
78     int cols = getCols(filename); // armazena o n mero de colunas da matriz
79     float *matrix = readFromFile(filename, rows, cols); // chama a fun   o
       readfromfile e l   a matriz do arquivo
80
81     //************************************ Defini   es manuais
       ************************************
82
83     int WindowSize = 7;  // Defina o tamanho da janela
84     float Threshold = 3.3; // Defina o limiar de decis o
85     int MinRange = 0; // Defina o valor m nimo do dom nio range [km]
86     int MaxRange = 1e5; // Defina o valor m ximo do dom nio range [km]
87   int vetorSize = 0;  // p/ guardar tamanho do vetor
88
89     //*************************************** IMPRESS ES
       ***************************************
90
91     // 1. Matriz mag
92   #if 0
93   printf("Matrix (%d x %d):\n", rows, cols);
94   printMatrix(matrix, rows, cols);
95   #endif
96
97   //*************************************************
98     // 2. Janela p/ cada caso de xFlag
99     // imprime os valores p/ todos casos de xflag
100    #if 0
```

```
  // Caso xFlag = -1: Impress o da matriz
  imprimirxFlag_minus1(WindowSize, vetorSize);
// Caso xFlag = -2: Impress o da     DeltaTransposta
imprimirxFlag_minus2(WindowSize, vetorSize);
// Caso xFlag = 1: Transformar     em vetor e exclui os termos 8192
  imprimirxFlag_1(WindowSize, vetorSize);
// Caso xFlag = 0: Transformar     Transposta e exclui os termos 8192
imprimirxFlag_0(WindowSize, vetorSize);
#endif


//*****************************************************
//3.  CFAR_CA_2D
  int* deltax = (int *)CFARWindow(WindowSize, 1, &vetorSize);    //
 Chamada para CFARWindow com xFlag = 1 para obter deltax
  int* deltay = (int *)CFARWindow(WindowSize, 0, &vetorSize);    //
 Chamada para CFARWindow com xFlag = 0 para obter deltay
  int *RangeHits = 0;
int *DopplerHits = 0;
  int NumHits = CFAR_CA_2D(matrix, Threshold, WindowSize, MinRange,
 MaxRange, filename, deltax, deltay, vetorSize, &RangeHits, &DopplerHits);
#if 0
  // Imprimir os resultados de RangeHits e DopplerHits
  printf("NumHits: %d\n", NumHits);
  printf("Range =           Doppler =\n");
  for (int i = 0; i < NumHits; i++) {
      printf("%11d      %15d\n", RangeHits[i], DopplerHits[i]);
  }
#endif

#if 0
  // Imprimir os resultados de RangeHits
  printf("Resultados de RangeHits:\n");
  for (int i = 0; i < NumHits; i++) {
      printf("%d\n", RangeHits[i]);
  }

  // Imprimir os resultados de DopplerHits
  printf("Resultados de DopplerHits:\n");
  for (int i = 0; i < NumHits; i++) {
      printf("%d\n", DopplerHits[i]);
  }
#endif

free(deltax);
  free(deltay);

  //*****************************************************
// 4. CFAR MultipleDopplerHitFilter e MultipleDopplerHitFilter
#if 0
int *F1Range = 0;
int *F1Doppler = 0;
int F1Resultado = MultipleDopplerHitFilter(matrix, DopplerHits, RangeHits,
  NumHits, filename, &F1Range, &F1Doppler);


// Imprimir os resultados de RangeHits e DopplerHits ap s o filtro 1
printf("MultipleDopplerFilterHits: \n");
printf("F1Range =           F1Doppler =\n");
for (int i = 0; i < F1Resultado; i++) {
```

```
157         printf("%11d      %15d\n", F1Range[i], F1Doppler[i]);
158     }
159 
160     free(F1Range);
161       free(F1Doppler);
162 
163     #endif
164 
165     #if 0
166       int *F2Range = 0;
167       int *F2Doppler = 0;
168       int F2Resultado = MultipleRangeHitFilter(matrix, DopplerHits, RangeHits,
          NumHits, filename, &F2Range, &F2Doppler);
169 
170 
171       // Imprimir os resultados de RangeHits e DopplerHits ap s o filtro 2
172       printf("MultipleRangeFilterHits: \n");
173       printf("F2Range =           F2Doppler =\n");
174       for (int i = 0; i < F2Resultado; i++) {
175           printf("%11d      %15d\n", F2Doppler[i], F2Range[i]);
176       }
177 
178       // Liberar mem ria alocada
179       free(F2Range);
180       free(F2Doppler);
181     #endif
182 
183 
184 
185 
186 
187       // Liberar mem ria alocada para os vetores
188       free(RangeHits);
189       free(DopplerHits);
190 
191       // Liberar a mem ria alocada para a matriz
192       free(matrix);
193       return 0;
194 }
195 
196 /*************************************************************************
197 * FUNC: void Quit(ERR err, char *name)
198 *
199 * DESC: Prints Error message and quits the program.
200 *************************************************************************/
201 void Quit(ERR err, const char *name)
202 {
203     static const char *ErrMess[] = {
204     "No error",
205     "Not enough memory",
206     "Unable to find the file",
207     "Error reading file",
208     "Unable to open the file",
209     "Error writing to file",
210     "Unable to attach buffer to file",
211     "Specified size doesn't match data size in file",
212       "Exceeded maximum number of iterations in function",
213       "Logic error in function",
214       "Incorrect command line argument",
215       "Invalid matrix dimensions in file",
```

```
216     "Incompatible number of rows between input matrices",
217     "Incompatible number of columns between input matrices",
218   "Unknown error"
219     };
220
221   system("cls"); // Limpa a tela (Windows)
222
223   if (err != NO_ERROR) {
224     if (name[0]) {
225       fprintf(stderr, "%s %s!\n", ErrMess[err], name);
226     } else {
227       fprintf(stderr, "%s!\n", ErrMess[err]);
228     }
229     exit(err);
230   } else {
231     exit(0); // Sa da sem erro
232   }
233 }
234
235
236 /*************************************************************************
237 * IMPLEMENTA  O DAS FUN   ES MATRIZ_ARD
238 *************************************************************************/
239
240 /*****************************************
241 * FUNC: getRows
242 * DESC: Conta o n mero de linhas em um arquivo.
243 * Par metro: filename - nome do arquivo
244 * Retorno: N mero de linhas no arquivo
245 *****************************************/
246 int getRows(const char *filename) {
247     FILE *file = fopen(filename, "r");
248     if (!file) {
249         Quit(READ_OPEN_ERR, filename);
250     }
251
252     int rows = 0;
253     int ch;
254
255     // Conta o n mero de linhas no arquivo
256     while ((ch = fgetc(file)) != EOF) {
257         if (ch == '\n') {
258             rows++;
259         }
260     }
261
262     fclose(file);
263     return rows;
264 }
265
266 /*****************************************
267 * FUNC: getCols
268 * DESC: Conta o n mero de colunas na primeira linha de um arquivo.
269 * Par metro: filename - nome do arquivo
270 * Retorno: N mero de colunas na primeira linha do arquivo
271 *****************************************/
272 int getCols(const char *filename) {
273     FILE *file = fopen(filename, "r");
274     if (!file) {
275         Quit(READ_OPEN_ERR, filename);
```

```
276     }
277 
278     int cols = 0;
279     int ch;
280 
281     // Conta o n mero de colunas na primeira linha
282     while ((ch = fgetc(file)) != EOF && ch != '\n') {
283         if (ch == ' ') {
284             cols++;
285         }
286     }
287 
288     fclose(file);
289     return cols + 1;
290 }
291 
292 /*****************************************
293 * FUNC: readFromFile
294 * DESC: L   uma matriz de valores float de um arquivo.
295 * Par metros: filename - nome do arquivo
296 *             rows - n mero de linhas na matriz
297 *             cols - n mero de colunas na matriz
298 * Retorno: Ponteiro para a matriz lida
299 *****************************************/
300 float *readFromFile(const char *filename, int rows, int cols) {
301     FILE *file = fopen(filename, "r");
302     if (!file) {
303         Quit(READ_OPEN_ERR, filename);
304     }
305 
306     float *matrix = (float *)malloc(rows * cols * sizeof(float));
307     if (!matrix) {
308         Quit(MEMORY_ERR, "for matrix allocation");
309     }
310 
311     // L   os valores da matriz do arquivo
312     for (int i = 0; i < rows; i++) {
313         for (int j = 0; j < cols; j++) {
314             if (fscanf(file, "%f", &matrix[i * cols + j]) != 1) {
315                 Quit(READ_ERR, "reading matrix element");
316             }
317         }
318     }
319 
320     fclose(file);
321     return matrix;
322 }
323 
324 /*****************************************
325 * FUNC: printMatrix
326 * DESC: Imprime uma matriz de valores float.
327 * Par metros: matrix - ponteiro para a matriz
328 *             rows - n mero de linhas na matriz
329 *             cols - n mero de colunas na matriz
330 *****************************************/
331 void printMatrix(float *matrix, int rows, int cols) {
332     // Imprime a matriz
333     for (int i = 0; i < rows; i++) {
334         for (int j = 0; j < cols; j++) {
335             printf("%f ", matrix[i * cols + j]);
```

```
        }
        printf("\n");
    }
}

/*****************************************************************************
* IMPLEMENTA  O DAS FUN  ES CFARWindow
*****************************************************************************/

/*****************************************
* FUNC: CFARWindow
* --------------------
* DESC: Cria uma janela CFAR (Constant False Alarm Rate) com base no tamanho
    fornecido.
*
* Par metros: WindowSize: O tamanho da janela, que deve ser um n mero
    mpar  entre 5 e 8192.
* xFlag: Um indicador utilizado para determinar o tipo de opera  o:
*        - xFlag = -2: Retorna a matriz transposta de  onde as c lulas
   de refer ncia armazenam valores de deslocamento delta do  ndice  y.
*                      As posi  es correspondentes  CUT e  s  c lulas
   do anel de guarda armazenam o valor 8192.
*        - xFlag = -1: Retorna a matriz  onde as c lulas de refer ncia
   armazenam valores de deslocamento delta do  ndice  x.
*                      As posi  es correspondentes  CUT e  s  c lulas
   do anel de guarda armazenam o valor 8192.
*        - xFlag = 1: Transforma a matriz  em um vetor e e retorna oos
   valores do vetor (as c lulas 8192 desaparecem).
*        - xFlag = 0: Transforma a matriz transposta de  em um vetor e
   retorna os valores do vetor transposto (as c lulas 8192 desaparecem).
* vetorSize: ponteiro para um inteiro que ser  preenchido com o tamanho do
   vetor retornado. Este par metro s  utilizado quando xFlag  0 ou 1.
*            Indica o n mero de elementos v lidos (diferentes de 8192) no
   vetor retornado.
*****************************************/
int** CFARWindow(int WindowSize, int xFlag, int* vetorSize) {
    // Verificar se o tamanho da janela  mpar  e est  no intervalo
   permitido
    if (WindowSize % 2 == 0 || WindowSize < 5 || WindowSize >= 8192) {
        return NULL;
    }

    int N = (WindowSize - 1) / 2;

    // Alocar matriz Delta
    int** Delta = (int**)malloc(WindowSize * sizeof(int*));
    for (int i = 0; i < WindowSize; i++) {
        Delta[i] = (int*)malloc(WindowSize * sizeof(int));
    }

    // Preencher a matriz Delta
    for (int col = 0; col < WindowSize; col++) {
        int v_col = -N + col;
        for (int row = 0; row < WindowSize; row++) {
            Delta[row][col] = v_col;
        }
    }

    // Ajustar o CUT e o anel de guarda
    for (int row = N - 1; row <= N + 1; row++) {
```

```
        for (int col = N - 1; col <= N + 1; col++) {
            Delta[row][col] = 8192;
        }
    }
    if (xFlag == -2) {
        // Retornar matriz transposta
        int** DeltaTransposta = (int**)malloc(WindowSize * sizeof(int*));
        for (int i = 0; i < WindowSize; i++) {
            DeltaTransposta[i] = (int*)malloc(WindowSize * sizeof(int));
        }
        for (int i = 0; i < WindowSize; i++) {
            for (int j = 0; j < WindowSize; j++) {
                DeltaTransposta[i][j] = Delta[j][i];
            }
        }
        return DeltaTransposta;
    } else if (xFlag == -1) {
        // Retornar matriz Delta
        return Delta;
    } else if (xFlag == 0 || xFlag == 1) {
        *vetorSize = 0;
        for (int i = 0; i < WindowSize; i++) {
            for (int j = 0; j < WindowSize; j++) {
                if (Delta[i][j] != 8192) {
                    (*vetorSize)++;
                }
            }
        }

        int* vetor = (int*)malloc(*vetorSize * sizeof(int));
        int index = 0;

        if (xFlag == 0) {
            // Transformar matriz transposta em vetor
            for (int i = 0; i < WindowSize; i++) {
                for (int j = 0; j < WindowSize; j++) {
                    if (Delta[j][i] != 8192) {
                        vetor[index++] = Delta[j][i];
                    }
                }
            }
        } else {
            // Transformar matriz em vetor
            for (int i = 0; i < WindowSize; i++) {
                for (int j = 0; j < WindowSize; j++) {
                    if (Delta[i][j] != 8192) {
                        vetor[index++] = Delta[i][j];
                    }
                }
            }
        }

        // Libera a mem ria alocada para a matriz Delta
        for (int i = 0; i < WindowSize; i++) {
            free(Delta[i]);
        }
        free(Delta);

        // Retornar o vetor
        return (int**)vetor;
```

```
445     } else {
446     // xFlag n o suportado
447     return NULL;
448   }
449 }
450 
451 /*****************************************
452 * FUNCTION: imprimirxFlag_minus1
453 * --------------------
454 * DESC: imprime a janela para xFlag = -1, retornando a matriz
455 *
456 * Par metros: WindowSize: O tamanho da janela, que deve ser um n mero
      mpar  entre 5 e 8192.
457 *         VetorSize: N o    utilizado
458 * OBS:
459 * xFlag: Um indicador utilizado para determinar o tipo de opera  o:
460 *        - xFlag = -2: Retorna a matriz transposta de     onde as c lulas
    de refer ncia armazenam valores de deslocamento delta do  ndice  y.
461 *                      As posi  es correspondentes    CUT e  s  c lulas
    do anel de guarda armazenam o valor 8192.
462 *        - xFlag = -1: Retorna a matriz     onde as c lulas de refer ncia
    armazenam valores de deslocamento delta do  ndice  x.
463 *                      As posi  es correspondentes    CUT e  s  c lulas
    do anel de guarda armazenam o valor 8192.
464 *        - xFlag = 1: Transforma a matriz     em um vetor e e retorna oos
    valores do vetor (as c lulas 8192 desaparecem).
465 *        - xFlag = 0: Transforma a matriz transposta de     em um vetor e
    retorna os valores do vetor transposto (as c lulas 8192 desaparecem).
466 *
467 *****************************************/
468 void imprimirxFlag_minus1(int WindowSize, int vetorSize) {
469     // Obter a matriz Delta usando CFARWindow com xFlag = -1
470     int **Delta = CFARWindow(WindowSize, -1, &vetorSize);
471 
472     printf("CFARWindow(%d, %d):\n", WindowSize, -1);
473     if (Delta != NULL) {
474         // Imprimir a matriz Delta
475         for (int i = 0; i < WindowSize; i++) {
476             for (int j = 0; j < WindowSize; j++) {
477                 printf("%5d ", Delta[i][j]);
478             }
479             printf("\n");
480         }
481 
482         // Liberar mem ria alocada para Delta
483         for (int i = 0; i < WindowSize; i++) {
484             free(Delta[i]);
485         }
486         free(Delta);
487     } else {
488         printf("Erro ao criar a janela CFAR para xFlag = -1.\n");
489     }
490 
491     printf("\n");
492 }
493 
494 /*****************************************
495 * FUNCTION: imprimirxFlag_minus2
496 * --------------------
497 * DESC: imprime a janela para xFlag = -2, retornando a matriz     transposta
```

```
498 *
499 * Par metros: WindowSize: O tamanho da janela, que deve ser um n mero
        mpar  entre 5 e 8192.
500 *         VetorSize: N o    utilizado
501 * OBS:
502 * xFlag: Um indicador utilizado para determinar o tipo de opera  o:
503 *        - xFlag = -2: Retorna a matriz transposta de     onde as c lulas
       de refer ncia armazenam valores de deslocamento delta do  ndice  y.
504 *                      As posi  es correspondentes    CUT e  s  c lulas
       do anel de guarda armazenam o valor 8192.
505 *        - xFlag = -1: Retorna a matriz     onde as c lulas de refer ncia
       armazenam valores de deslocamento delta do  ndice  x.
506 *                      As posi  es correspondentes    CUT e  s  c lulas
       do anel de guarda armazenam o valor 8192.
507 *        - xFlag = 1: Transforma a matriz     em um vetor e e retorna oos
       valores do vetor (as c lulas 8192 desaparecem).
508 *        - xFlag = 0: Transforma a matriz transposta de     em um vetor e
       retorna os valores do vetor transposto (as c lulas 8192 desaparecem).
509 *
510 ******************************************/
511 void imprimirxFlag_minus2(int WindowSize, int vetorSize) {
512     // Obter a matriz DeltaTransposta usando CFARWindow com xFlag = -2
513     int **DeltaTransposta = CFARWindow(WindowSize, -2, &vetorSize);
514 
515     printf("CFARWindow(%d, %d):\n", WindowSize, -2);
516     if (DeltaTransposta != NULL) {
517         // Imprimir a matriz DeltaTransposta
518         for (int i = 0; i < WindowSize; i++) {
519             for (int j = 0; j < WindowSize; j++) {
520                 printf("%5d ", DeltaTransposta[i][j]);
521             }
522             printf("\n");
523         }
524 
525         // Liberar mem ria alocada para DeltaTransposta
526         for (int i = 0; i < WindowSize; i++) {
527             free(DeltaTransposta[i]);
528         }
529         free(DeltaTransposta);
530     } else {
531         printf("Erro ao criar a janela CFAR para xFlag = -2.\n");
532     }
533 
534     printf("\n");
535 }
536 
537 /*****************************************
538 * FUNCTION: imprimirxFlag_1
539 * --------------------
540 * DESC: xFlag = 1, transforma a matriz     em um vetor e imprime o vetor
541 *
542 * Par metros: WindowSize: O tamanho da janela, que deve ser um n mero
        mpar  entre 5 e 8192.
543 *         VetorSize: ponteiro para um inteiro que ser  preenchido com o
       tamanho do vetor retornado.
544 * OBS:
545 * xFlag: Um indicador utilizado para determinar o tipo de opera  o:
546 *        - xFlag = -2: Retorna a matriz transposta de     onde as c lulas
       de refer ncia armazenam valores de deslocamento delta do  ndice  y.
547 *                      As posi  es correspondentes    CUT e  s  c lulas
```

```
    do anel de guarda armazenam o valor 8192.
548 *         - xFlag = -1: Retorna a matriz     onde as c lulas de refer ncia
    armazenam valores de deslocamento delta do  ndice  x.
549 *                       As posi  es correspondentes    CUT e  s  c lulas
    do anel de guarda armazenam o valor 8192.
550 *         - xFlag = 1: Transforma a matriz     em um vetor e e retorna oos
    valores do vetor (as c lulas 8192 desaparecem).
551 *         - xFlag = 0: Transforma a matriz transposta de     em um vetor e
    retorna os valores do vetor transposto (as c lulas 8192 desaparecem).
552 *
553 *****************************************/
554 void imprimirxFlag_1(int WindowSize, int vetorSize) {
555     // Obter o vetor Delta usando CFARWindow com xFlag = 1
556     int *vetorDelta = (int *)CFARWindow(WindowSize, 1, &vetorSize);
557
558     printf("CFARWindow(%d, %d):\n", WindowSize, 1);
559     if (vetorDelta != NULL) {
560         // Imprimir o vetor Delta
561         printVetor(vetorDelta, vetorSize);
562
563         // Liberar mem ria alocada para vetorDelta
564         free(vetorDelta);
565     } else {
566         printf("Erro ao criar a janela CFAR para xFlag = 1.\n");
567     }
568
569     printf("\n");
570 }
571
572 /*****************************************
573 * FUNCTION: imprimirxFlag_0
574 * --------------------
575 * DESC: xFlag = 0, transforma a matriz     transposta em um vetor e imprime
    o vetor
576 *
577 * Par metros: WindowSize: O tamanho da janela, que deve ser um n mero
     mpar  entre 5 e 8192.
578 *          VetorSize: ponteiro para um inteiro que ser  preenchido com o
    tamanho do vetor retornado.
579 * OBS:
580 * xFlag: Um indicador utilizado para determinar o tipo de opera  o:
581 *         - xFlag = -2: Retorna a matriz transposta de     onde as c lulas
    de refer ncia armazenam valores de deslocamento delta do  ndice  y.
582 *                       As posi  es correspondentes    CUT e  s  c lulas
    do anel de guarda armazenam o valor 8192.
583 *         - xFlag = -1: Retorna a matriz     onde as c lulas de refer ncia
    armazenam valores de deslocamento delta do  ndice  x.
584 *                       As posi  es correspondentes    CUT e  s  c lulas
    do anel de guarda armazenam o valor 8192.
585 *         - xFlag = 1: Transforma a matriz     em um vetor e e retorna oos
    valores do vetor (as c lulas 8192 desaparecem).
586 *         - xFlag = 0: Transforma a matriz transposta de     em um vetor e
    retorna os valores do vetor transposto (as c lulas 8192 desaparecem).
587 *
588 *****************************************/
589 void imprimirxFlag_0(int WindowSize, int vetorSize) {
590     // Obter o vetor usando CFARWindow com xFlag = 0
591     int *vetorDeltaTransposta = (int *)CFARWindow(WindowSize, 0, &vetorSize)
    ;
592
```

```
    printf("CFARWindow(%d, %d):\n", WindowSize, 0);
    if (vetorDeltaTransposta != NULL) {
        // Imprimir o vetor
        printVetor(vetorDeltaTransposta, vetorSize);

        // Liberar mem ria alocada para vetorDeltaTransposta
        free(vetorDeltaTransposta);
    } else {
        printf("Erro ao criar a janela CFAR para xFlag = 0.\n");
    }

    printf("\n");
}

/*****************************************************
* IMPLEMENTA  O DAS FUN   O PRINTVETOR
*****************************************************/
/*****************************************
* FUNCTION: printVetor
* --------------------
* DESC: Imprime os valores de um vetor.
*
* Par metros: vetor: Ponteiro para o vetor a ser impresso.
*         size: Tamanho do vetor.
* retorno: impress o do vetor
*****************************************/
void printVetor(int* vetor, int size) {
    // Verifica se o ponteiro    nulo
    if (vetor == NULL) {
        Quit(LOGIC_ERR, "Null pointer passed to printVetor");
    }

    // Imprime os valores do vetor
    for (int i = 0; i < size; i++) {
        printf("%5d ", vetor[i]);
    }
    printf("\n");
}

/*****************************************************
* IMPLEMENTA  O DA FUNC CFAR_CA_2D
*****************************************************/
/*****************************************
* FUNC: CFAR_CA_2D
* --------------------
* DESC:  Implementa  o da fun  o CFAR_CA_2D para detec  o de alvos em
   matrizes 2D de magnitude.
*
* Par metros:
*     mag: Matriz de magnitude    no dom nio Doppler e Range, de dimens o
   [NumDoppler, NumRangePoints].
*     Threshold: Valor em ponto flutuante na faixa [2.0, 20.0] que define o
   limiar de decis o do CFAR.
*     WindowSize: Tamanho da janela de CFAR no dom nio Range. WindowSize
   deve ser um inteiro  mpar  igual ou maior que 5.
*     MinRange e MaxRange: Delimita  o do dom nio Range no mov da janela
   W(range,v) sobre o dominio (range,v) de   .
*     filename: chama o arquivo filename e aplica as fun  es auxiliares
*   deltax e deltay: - (deltax) xFlag = 1: Transforma a matriz     em um
   vetor e e retorna oos valores do vetor.
```

```
647 *                - (deltay) xFlag = 0: Transforma a matriz transposta de
    em um vetor e retorna os valores do vetor transposto.
648 *   vetorSize: Tamanho dos vetores deltax e deltay, que representa o n mero
    de elementos nesses vetores.
649 * Nota: a fun   o CFAR_CA_2D() chama a fun   o CFARWindow(), cujo c digo
    foi descrito acima.
650 *
651 * RETORNO DA FUNC:
652 *   LinHit(RangeHits): Vetor contendo os   ndices   dos hits do CFAR no
    dom nio Doppler v da matriz   .
653 *   ColHit(DopplerHits): Vetor contendo os   ndices   dos hits do CFAR no
    dom nio Range da matriz   .
654 *   NumHits: Indica o n mero total de hits resultantes do CFAR
655
656 * Nota: LinHit(RangeHits) e ColHit(DopplerHits) s o vetores com o mesmo
    n mero NumHits de elementos,
657 ****************************************/
658 int CFAR_CA_2D(float *mag, float Threshold, int WindowSize, int MinRange,
    int MaxRange, const char *filename, int* deltax, int* deltay, int
    vetorSize, int **RangeHits, int **DopplerHits) {
659     int NRanges = getCols(filename);
660     int NFDoppl = getRows(filename);
661
662     MinRange = round(MinRange / (T * C));
663     MaxRange = round(MaxRange / (T * C));
664
665     int Ndelta = vetorSize;
666
667     // Inicializa  o das vari veis para armazenar os hits
668     int maxHits = 1000;  // N mero m ximo esperado de hits
669     int NumHits = 0;
670     *RangeHits = (int *)malloc(maxHits * sizeof(int));
671     *DopplerHits = (int *)malloc(maxHits * sizeof(int));
672
673     // Verificar se a aloca  o de mem ria foi bem-sucedida
674     if (*RangeHits == NULL || *DopplerHits == NULL) {
675         Quit(MEMORY_ERR, "for hit vectors allocation");
676     }
677
678     // Algoritmo CFAR_CA_2D para detec  o de hits
679     for (int row = WindowSize; row < NFDoppl - WindowSize; row++) {
680         for (int col = WindowSize; col < NRanges - WindowSize; col++) {
681             if (col < MinRange || col > MaxRange) {
682                 continue;
683             }
684
685             float u = 0.0;
686             for (int n = 0; n < Ndelta; n++) {
687                 u += (1.0 / Ndelta) * mag[((row + deltay[n]) * NRanges) + (
    col + deltax[n])];
688             }
689
690             if (mag[row * NRanges + col] > u * Threshold) {
691                 // Verificar se h   espa o suficiente nos vetores
692                 if (NumHits >= maxHits) {
693                     // Realocar os vetores para aumentar a capacidade
694                     maxHits *= 2;  // Dobrar a capacidade de hits m ximos
695                     *RangeHits = (int *)realloc(*RangeHits, maxHits * sizeof
    (int));
696                     *DopplerHits = (int *)realloc(*DopplerHits, maxHits *
```

```
    sizeof(int));
                    if (*RangeHits == NULL || *DopplerHits == NULL) {
                        free(*RangeHits);
                        free(*DopplerHits);
                        Quit(MEMORY_ERR, "for hit vectors reallocation");
                    }
                }

                // Armazenar os hits detectados
                (*RangeHits)[NumHits] = col;
                (*DopplerHits)[NumHits] = row;
                NumHits++;
            }
        }
    }
    return NumHits; // Retorna o n mero de hits encontrados
}

/*****************************************************
* IMPLEMENTA  O DAS FUN  ES DOS FILTROS
*****************************************************/
/*****************************************
*FUNC: MultipleDopplerHitFilter
*--------------------
*DESC: Implementa  o da fun  o MultipleDopplerHitFilter para filtrar hits Doppler em matrizes de magnitude.
* Par metros:
*   mag: Ponteiro para a matriz de magnitude.
*   DopplerHits: Vetor de hits Doppler.
*   RangeHits: Vetor de hits de range.
*   numHits: N mero total de hits n o filtrados.
*   filename: Nome do arquivo usado para obter o n mero de colunas da matriz de magnitude.
*   F1Range: Ponteiro para a matriz filtrada de hits de range.
*   F1Doppler: Ponteiro para a matriz filtrada de hits Doppler.
*
* Retornos:
*   F1Range:  Matriz filtrada contendo os hits de alcance.
*   F1Doppler: Matriz filtrada contendo os hits Doppler
*   numValidResultsF1: N mero de resultados v lidos ap s a filtragem.
*****************************************/
int MultipleDopplerHitFilter(float *mag, int *DopplerHits, int *RangeHits, int numHits, const char *filename, int **F1Range, int **F1Doppler) {
    // n mero total de hits n o filtrados
    int NUnfilteredHits = numHits;

    // indices para a linha (GrIdx) e coluna (ElIdx) durante a itera  o
    int ElIdx = 0;
    int GrIdx = 0;

    // n mero de colunas na matriz de magnitude (mag) obtido da fun  o getCols
    int Cols = getCols(filename);

    // primeira passagem para calcular o n mero de linhas (GrIdx) e o n mero m ximo de colunas (ElIdx)
    for (int n = 0; n <= NUnfilteredHits - 2; n++) {
        // se o valor de DopplerHits no  ndice  atual for igual ao pr ximo, incrementa o  ndice  da coluna
        if (DopplerHits[n] == DopplerHits[n + 1]) {
```

```
750             ElIdx++;
751         } else {
752             // se o valor de DopplerHits mudar, incrementa o  ndice  da
    linha e reseta o  ndice  da coluna
753             GrIdx++;
754             ElIdx = 0;
755         }
756     }
757     // incrementar GrIdx para incluir o  ltimo  grupo
758     GrIdx++;
759 
760     // armazenar os valores de GrIdx e ElIdx como n mero de linhas e
    colunas para as matrizes F1Range e F1Doppler
761     int linhas = GrIdx;
762     int colunas = ElIdx + 1; // Adiciona 1 porque o  ndice  come a em 0
763 
764     // alocar mem ria para F1Range e F1Doppler com os tamanhos calculados
765     *F1Range = (int *)calloc(linhas * colunas, sizeof(int));
766     *F1Doppler = (int *)calloc(linhas * colunas, sizeof(int));
767 
768     // verificar se a aloca  o de mem ria foi bem-sucedida
769     if (*F1Range == NULL || *F1Doppler == NULL) {
770         Quit(MEMORY_ERR, "for hit vectors allocation");
771     }
772 
773     // redefinir os  ndices  para preencher as matrizes com os dados
774     ElIdx = 0;
775     GrIdx = 0;
776 
777     // preencher F1Range e F1Doppler com os valores dos hits
778     for (int n = 0; n <= NUnfilteredHits - 2; n++) {
779           (*F1Range)[GrIdx * colunas + ElIdx] = RangeHits[n];
780           (*F1Doppler)[GrIdx * colunas + ElIdx] = DopplerHits[n];
781           // se o valor de DopplerHits mudar, incrementa o  ndice  da linha
    e reseta o  ndice  da coluna
782           if (DopplerHits[n] == DopplerHits[n + 1]) {
783               ElIdx++;
784         } else {
785             GrIdx++;
786             ElIdx = 0;
787         }
788     }
789 
790     // copiando o  ltimo  valor do array para F1Range e F1Doppler
791     (*F1Range)[GrIdx * colunas + ElIdx] = RangeHits[NUnfilteredHits - 1];
792     (*F1Doppler)[GrIdx * colunas + ElIdx] = DopplerHits[NUnfilteredHits -
    1];
793 
794     // p/ alocar mem ria para armazenar os m ximos encontrados
795     int *F1_RangeHits = (int *)calloc(linhas, sizeof(int));
796     int *F1_DopplerHits = (int *)calloc(linhas, sizeof(int));
797 
798     // p/ verificar se a aloca  o de mem ria foi bem-sucedida
799     if (F1_RangeHits == NULL || F1_DopplerHits == NULL) {
800         Quit(MEMORY_ERR, "for hit vectors allocation");
801     }
802 
803     // encontrando os m ximos valores para cada linha (grupo)
804     for (GrIdx = 0; GrIdx <= linhas-1; GrIdx++) {
805         float maxval = 0.0; // Inicializa o valor m ximo como zero
```

```
806         unsigned IdxMax = 0; // Inicializa o  ndice  do valor m ximo
807 
808         // Percorrer as colunas da linha atual para encontrar o valor
      m ximo
809         for (ElIdx = 0; ElIdx <= colunas-1; ElIdx++) {
810             // Verificar se o valor de DopplerHits n o    zero
811             if ((*F1Doppler)[GrIdx * colunas + ElIdx] != 0) {
812                 int Idx1 = (*F1Range)[GrIdx * colunas + ElIdx];
813                 int Idx2 = (*F1Doppler)[GrIdx * colunas + ElIdx];
814 
815                 // Atualizar o valor m ximo e o  ndice  do valor m ximo
816                 if (mag[Idx2 * Cols + Idx1] > maxval) {
817                     maxval = mag[Idx2 * Cols + Idx1];
818                     IdxMax = ElIdx;
819                 }
820             }
821         }
822         // P/ armazenar os valores m ximos encontrados para a linha atual
823         F1_RangeHits[GrIdx] = (*F1Range)[GrIdx * colunas + IdxMax];
824         F1_DopplerHits[GrIdx] = (*F1Doppler)[GrIdx * colunas + IdxMax];
825     }
826 
827   // p/ adicionar a vari vel para armazenar o n mero de resultados
      v lidos
828     int numValidResultsF1 = linhas;
829     // Liberar a mem ria  tempor ria utilizada para armazenar os dados
      intermedi rios
830     free(*F1Range);
831     free(*F1Doppler);
832 
833 
834 
835     // p/ atribuir os resultados finais para F1Range e F1Doppler
836     *F1Range = F1_RangeHits;
837     *F1Doppler = F1_DopplerHits;
838     #if 0
839     // Imprimir valores finais de F1Range e F1Doppler
840     printf("F1Range final:\n");
841     for (int i = 0; i < numValidResultsF1; i++) {
842         printf("%d ", (*F1Range)[i]);
843     }
844     printf("\n");
845 
846     printf("F1Doppler final:\n");
847     for (int i = 0; i < numValidResultsF1; i++) {
848         printf("%d ", (*F1Doppler)[i]);
849     }
850     printf("\n");
851     #endif
852     return numValidResultsF1;
853 }
854 
855 
856 
857 
858 
859 /*****************************************
860 *FUNC: MultipleRangeHitFilter
861 *--------------------
862 *DESC: Implementa  o da fun  o MultipleRangeHitFilter para filtrar hits
```

```
   de alcance em matrizes de magnitude.
* Par metros:
*   mag: Ponteiro para a matriz de magnitude.
*   RangeHits: Vetor de hits de alcance.
*   DopplerHits: Vetor de hits Doppler.
*   numHits: N mero total de hits n o filtrados.
*   filename: Nome do arquivo usado para obter o n mero de colunas da
   matriz de magnitude.
*   F2Range: Ponteiro para a matriz filtrada de hits de alcance.
*   F2Doppler: Ponteiro para a matriz filtrada de hits Doppler.
*
* Retornos:
*   F2Range:  Matriz filtrada contendo os hits de alcance.
*   F2Doppler: Matriz filtrada contendo os hits Doppler.
*   numValidResultsF2: N mero de resultados v lidos ap s a filtragem.
*****************************************/
/**
int MultipleRangeHitFilter(float *mag, int *RangeHits, int *DopplerHits, int
    numHits, const char *filename, int **F2Range, int **F2Doppler) {
    // n mero total de hits n o filtrados
    int NUnfilteredHits = numHits;

    //  ndices  para a linha (GrIdx) e coluna (ElIdx) durante a itera  o
    int ElIdx = 0;
    int GrIdx = 0;

    // n mero de colunas na matriz de magnitude (mag) obtido da fun  o
   getCols
    int Cols = getCols(filename);
  #if 1
  // Imprimir valores de RangeHits e DopplerHits antes do filtro
    printf("RangeHits antes do filtro:\n");
    for (int i = 0; i < NUnfilteredHits; i++) {
        printf("%d ", RangeHits[i]);
    }
    printf("\n");

    printf("DopplerHits antes do filtro:\n");
    for (int i = 0; i < NUnfilteredHits; i++) {
        printf("%d ", DopplerHits[i]);
    }
    printf("\n");
    #endif

    // primeira passagem para calcular o n mero de linhas (GrIdx) e o
   n mero  m ximo de colunas (ElIdx)
    for (int n = 0; n <= NUnfilteredHits - 2; n++) {
        // se o valor de RangeHits no  ndice  atual for igual ao pr ximo,
   incrementa o  ndice  da coluna
        if (RangeHits[n] == RangeHits[n + 1]) {
            ElIdx++;
        } else {
            // se o valor de RangeHits mudar, incrementa o  ndice  da linha
   e reseta o  ndice  da coluna
            GrIdx++;
            ElIdx = 0;
        }
    }
    // incrementar GrIdx para incluir o  ltimo  grupo
    GrIdx++;
```

```
916
917     // armazenar os valores de GrIdx e ElIdx como n mero de linhas e
    colunas para as matrizes F2Range e F2Doppler
918     int linhas = GrIdx;
919     int colunas = ElIdx + 1; // Adiciona 1 porque o  ndice  come a em 0
920
921     // alocar mem ria para F2Range e F2Doppler com os tamanhos calculados
922     *F2Range = (int *)calloc(linhas * colunas, sizeof(int));
923     *F2Doppler = (int *)calloc(linhas * colunas, sizeof(int));
924
925     // verificar se a aloca  o de mem ria foi bem-sucedida
926     if (*F2Range == NULL || *F2Doppler == NULL) {
927         Quit(MEMORY_ERR, "for hit vectors allocation");
928     }
929
930     // redefinir os  ndices  para preencher as matrizes com os dados
931     ElIdx = 0;
932     GrIdx = 0;
933
934     // preencher F2Range e F2Doppler com os valores dos hits
935     for (int n = 0; n <= NUnfilteredHits - 2; n++) {
936         (*F2Range)[GrIdx * colunas + ElIdx] = RangeHits[n];
937         (*F2Doppler)[GrIdx * colunas + ElIdx] = DopplerHits[n];
938         // se o valor de RangeHits mudar, incrementa o  ndice  da linha e
    reseta o  ndice  da coluna
939         if (RangeHits[n] == RangeHits[n + 1]) {
940             ElIdx++;
941         } else {
942             GrIdx++;
943             ElIdx = 0;
944         }
945     }
946
947     // copiando o  ltimo  valor do array para F2Range e F2Doppler
948     (*F2Range)[GrIdx * linhas + ElIdx] = RangeHits[NUnfilteredHits - 1];
949     (*F2Doppler)[GrIdx * colunas + ElIdx] = DopplerHits[NUnfilteredHits -
    1];
950
951     // p/ alocar mem ria para armazenar os m ximos encontrados
952     int *F2_RangeHits = (int *)calloc(linhas, sizeof(int));
953     int *F2_DopplerHits = (int *)calloc(linhas, sizeof(int));
954
955     // p/ verificar se a aloca  o de mem ria foi bem-sucedida
956     if (F2_RangeHits == NULL || F2_DopplerHits == NULL) {
957         Quit(MEMORY_ERR, "for hit vectors allocation");
958     }
959
960     // encontrando os m ximos valores para cada linha (grupo)
961     for (GrIdx = 0; GrIdx <= (colunas - 1); GrIdx++) {
962         float maxval = 0.0; // Inicializa o valor m ximo como zero
963         unsigned IdxMax = 0; // Inicializa o  ndice  do valor m ximo
964
965         // Percorrer as colunas da linha atual para encontrar o valor
    m ximo
966         for (ElIdx = 0; ElIdx <= (linhas - 1); ElIdx++) {
967             // Verificar se o valor de RangeHits n o    zero
968             if ((*F2Range)[GrIdx * colunas + ElIdx] != 0) {
969                 int Idx1 = (*F2Range)[GrIdx * colunas + ElIdx];
970                 int Idx2 = (*F2Doppler)[GrIdx * colunas + ElIdx];
971
```

```
                // Atualizar o valor m ximo e o  ndice  do valor m ximo
                if (mag[Idx1 * Cols + Idx2] > maxval) {
                    maxval = mag[Idx2 * Cols + Idx1];
                    IdxMax = ElIdx;
                }
            }
        }
        // P/ armazenar os valores m ximos encontrados para a linha atual
        F2_RangeHits[GrIdx] = (*F2Range)[GrIdx * colunas + IdxMax];
        F2_DopplerHits[GrIdx] = (*F2Doppler)[GrIdx * colunas + IdxMax];
    }

    // p/ adicionar a vari vel para armazenar o n mero de resultados
    v lidos
    int numValidResultsF2 = colunas;
    // Liberar a mem ria tempor ria utilizada para armazenar os dados
    intermedi rios
    free(*F2Range);
    free(*F2Doppler);

    // p/ atribuir os resultados finais para F2Range e F2Doppler
    *F2Range = F2_RangeHits;
    *F2Doppler = F2_DopplerHits;

    #if 1
    // Imprimir valores finais de F2Range e F2Doppler
    printf("F2Range final:\n");
    for (int i = 0; i < numValidResultsF2; i++) {
        printf("%d ", (*F2Range)[i]);
    }
    printf("\n");

    printf("F2Doppler final:\n");
    for (int i = 0; i < numValidResultsF2; i++) {
        printf("%d ", (*F2Doppler)[i]);
    }
    printf("\n");
    #endif

    return numValidResultsF2;
}

*/
```

Referências

[1] *Técnicas de Radar*, https://www.fccdecastro.com.br/pdf/TR$_{C}apIII.pdf$